Never Assuming that Your Secrets Are Safe

Sean Barnum, Cigital, Inc. [vita[4]]
Michael Gegick, Cigital, Inc. [vita[5]]

2005-09-14

Relying on an obscure design or implementation does not guarantee that a system is secured. You should always assume that an attacker can obtain enough information about your system to launch an attack. Tools such as decompilers and disassemblers allow attackers to obtain sensitive information that may be stored in binary files. Also, inside attacks, which may be accidental or malicious, can lead to security exploits. Using real protection mechanisms to secure sensitive information should be the ultimate means of protecting your secrets.

Detailed Description Excerpts

According to Howard and LeBlanc [Howard 02] in Chapter 3, "Security Principles to Live By," in "Never Depend on Security Through Obscurity Alone" from pages 66-67:

> Always assume that an attacker knows everything that you know -- assume the attacker has access to all source code and all designs. Even if this is not true, it is trivially easy for an attacker to determine obscured information. Other parts of this book show many examples of how such information can be found. Obscurity is a useful defense, so long as it is not your only defense. In other words, it's quite valid to use obscurity as a small part of an overall defense in depth strategy.

We include two relevant principles from Viega and McGraw [Viega 02]. In Chapter 5, "Guiding Principles for Software Security," in "Principle 8: Remember that Hiding Secrets Is Hard" from pages 109-111:[9]

> Security is often about keeping secrets. Users don't want their personal data leaked. Keys must be kept secret to avoid eavesdropping and tampering. Top-secret algorithms need to be protected from competitors. These kinds of requirements are almost always high on the list, but turn out to be far more difficult to meet than the average user may suspect.
>
> Many people make an implicit assumption that secrets in a binary are likely to stay secret, maybe because it seems very difficult to extract secrets from a binary. It is true that binaries are complex. However, as we discuss in Chapter 4 [in *Building Secure Software*], keeping the "secrets" secret in a binary is incredibly difficult. One problem is that some attackers are surprisingly good at reverse engineering binaries. They can pull a binary apart, and figure out what it does. The transparent binary problem is why copy protection schemes for software tend not to actually work. Skilled youths will circumvent any protection that a company tries to hardcode into their software, and release "cracked" copies. For years, there was an arms race and an associated escalation in techniques of both sides; vendors would try harder to keep people from finding the secrets to "unlock" software, and the software crackers would try harder to break the software. For the most part, the crackers won. Cracks for interesting software like DVD viewers or audio players tend to

4. daisy:35 (Barnum, Sean)
5. daisy:345 (Gegick, Michael)
9.

show up on the same day that the software is officially released, and sometimes sooner.

It may appear that software running server-side on a dedicated network could keep secrets safe, but that's not necessarily the case. Avoid trusting even a dedicated network if possible. Think through a scenario in which some unanticipated flaw lets an intruder steal your software. This actually happened to Id software right before they released the first version of Quake.

Even the most secure networks are often amenable to insider attacks. Several studies show that the most common threat to companies is the insider attack, where a disgruntled employee abuses access. Sometimes the employee isn't even disgruntled; maybe he just takes his job home, and a friend goes prodding around where she shouldn't. Think about the fact that many companies are not able to protect their firewall-guarded software from a malicious janitor. If someone is really intent on getting to software through illegal means, it can probably be done. When we point out the possibility of an insider attack to clients, we often hear "That won't happen to us; we trust our employees." But relying on this reasoning is dangerous even though 95% of the people we talk to say the same thing. Given that most attacks are perpetrated by insiders, there's a large logical gap here, suggesting that most of the people who believe they can trust their employees must be wrong. Remember that employees may like your environment, but when it comes down to it, most of them have a business relationship with your company, not a personal relationship. The moral here is that it pays to be paranoid.

The infamous FBI spy Richard P. Hanssen carried out the ultimate insider attack against U.S. classified networks for over 15 years. Hanssen was assigned to the FBI counterintelligence squad in 1985, around the same time he became a traitor to his country. During some of that time, he was root on a UNIX system. The really disturbing thing is that Hanssen created code (in C and Pascal) that was (is?) used to carry out various communications functions in the FBI. Apparently he wrote code used by agents in the field to cable back to the home office. We sincerely hope that any and all code used in such critical functions is carefully checked before it becomes widely used. If not, the possibility that a Trojan is installed in the FBI comm system is extremely high. Any and all code that was ever touched by Hanssen needs to be checked. In fact, Hanssen sounds like the type of person who might even be able to create hard-to-detect distributed attacks that use covert channels to leak information out.

Software is a powerful tool, both for good and for evil. Since most people treat software as "magic" and never actually look at its inner-workings, the potential for serious misuse and abuse is a very real risk.

Keeping secrets is hard, and is almost always a source of security risk.

In Chapter 5, "Guiding Principles for Software Security," in "Principle 10: Use Your Community Resources" from pages 112-113:

While it's not a good idea to follow the herd blindly, there is something to be said for strength in numbers. Repeated use without failure promotes trust. Public scrutiny does as well.

For example, in the cryptography field it is considered a bad idea to trust any algorithm that isn't public knowledge and hasn't been widely scrutinized. There's no real solid mathematical proof of the security of most cryptographic algorithms; they're trusted only when a large enough number of smart people have spent a lot of time trying to break them, and all fail to make substantial progress.

Many developers find it exciting to write their own cryptographic algorithms, sometimes banking on the fact that if they are weak, security by obscurity will help them. Repeatedly, these sorts of hopes are dashed on the rocks (for two examples, recall the Netscape and E*Trade breaks mentioned above). The argument generally goes that a secret algorithm is better than a publicly known one. We've already discussed why it is not a good idea to expect any algorithm to stay

secret for very long. The RC2 and RC4 encryption algorithms, for example, were supposed to be RSA Security trade secrets. However, they were both ultimately reverse engineered and posted anonymously to the Internet.

In any case, cryptographers design their algorithms so that knowledge of the algorithm is unimportant to its security properties. Good cryptographic algorithms work because they rely on keeping a small piece of data secret (the key), not because the algorithm itself is secret. That is, the only thing a user needs to keep private is the key. If a user can do that, and the algorithm is actually good (and the key is long enough), then even an attacker intimately familiar with the algorithm will be unable to break the crypto (given reasonable computational resources).

Similarly, it's far better to trust security libraries that have been widely used, and widely scrutinized. Of course, they might contain bugs that haven't been found. But at least it is possible to leverage the experience of others.

This principle only applies if you have reason to believe that the community is doing its part to promote the security of the components you want to use. As we discussed at length in Chapter 4 [in *Building Secure Software*], one common fallacy is to believe that "Open Source" software is highly likely to be secure, because source availability will lead to people performing security audits. There's strong evidence to suggest that source availability doesn't provide the strong incentive for people to review source code and design that many would like to believe exists. For example, many security bugs in widely used pieces of free software have gone unnoticed for years. In the case of the most popular FTP server around, several security bugs went unnoticed for over a decade!

References

[Howard 02]	Howard, Michael & LeBlanc, David. *Writing Secure Code*. 2nd ed. Redmond, WA: Microsoft Press, 2002.
[Viega 02]	Viega, John & McGraw, Gary. *Building Secure Software: How to Avoid Security Problems the Right Way*. Boston, MA: Addison-Wesley, 2002.

Cigital, Inc. Copyright

1. mailto:copyright@cigital.com

Velden

Naam	Waarde
Copyright Holder	Cigital, Inc.

Velden

Naam	Waarde
is-content-area-overview	false
Content Areas	Knowledge/Principles
SDLC Relevance	Design
Workflow State	Publishable